Cairo University
Faculty of Engineering
Computer Engineering Department

Data Structures and Algorithms
Sheet #4 – Stacks and Queues

Part I: Exercises

1. Imagine you have a stack of integers, S, and a queue of integers, Q. Draw a picture of S and Q after the following operations:

```
pushStack (S, 3)
pushStack (S, 12)
enqueue (Q, 5)
enqueue (Q, 8)
popStack (S, x)
pushStack (S, 2)
enqueue (Q, x)
dequeue (Q, y)
pushStack (S, x)
pushStack (S, y)
```

2. What would be the value of queues Q1 and Q2, and stack S after the following algorithm segment:

```
1  S  = createStack
2  Q1 = createQueue
3  Q2 = createQueue
4  enqueue (Q1, 5)
5  enqueue (Q1, 6)
6  enqueue (Q1, 9)
7  enqueue (Q1, 0)
8  enqueue (Q1, 7)
9  enqueue (Q1, 5)
10 enqueue (Q1, 0)
11 enqueue (Q1, 2)
12 enqueue (Q1, 6)
13 loop (not emptyQueue (Q1))
   1 dequeue (Q1, x)
   2 if (x == 0)
      1 z = 0
      2 loop (not emptyStack (S))
         1 popStack (S, y)
         2 z = z + y
      3 end loop
      4 enqueue (Q2, z)
   3 else
      1 pushStack (S, x)
   4 end if
14 end loop
```

3. What would be the contents of queue Q after the following code is executed and the following data are entered?

```
1  Q = createQueue
2  loop (not end of file)
   1 read number
   2 if (number not 0)
      1 enqueue (Q, number)
   3 else
      1 x = queuerear (Q)
      2 enqueue (Q, x)
   4 end if
3  end loop
```

The data are: 5, 7, 12, 4, 0, 4, 6, 8, 67, 34, 23, 5, 0, 44, 33, 22, 6, 0.

4. What would be the contents of queue Q1 and queue Q2 after the following code is executed and the following data are entered?

```
1 Q1 = createQueue
2 Q2 = createQueue
3 loop (not end of file)
   1 read number
   2 enqueue (Q1, number)
   3 enqueue (Q2, number)
   4 loop (not empty Q1)
      1 dequeue (Q1, x)
      2 enqueue (Q2, x)
   5 end loop
4 end loop
```

The data are 5, 7, 12, 4, 0, 4, 6.

5. What would be the contents of queue Q1 after the following code is executed and the following data are entered?

```
1 Q1 = createQueue
2 S1 = createStack
3 loop (not end of file)
   1 read number
   2 if (number not 0)
      1 pushStack (S1, number)
   3 else
      1 popStack (S1, x)
      2 popStack (S1, x)
      3 loop (not empty S1)
         1 popStack (S1, x)
         2 enqueue (Q1, x)
      4 end loop
   4 end if
4 end loop
```

The data are 5, 7, 12, 4, 0, 4, 6, 8, 67, 34, 23, 5, 0, 44, 33, 22, 6, 0.

6. Imagine that the contents of queue Q1 and queue Q2 are as shown. What would be the contents of Q3 after the following code is executed? The queue contents are shown front (left) to rear (right).

Q1: 42 30 41 31 19 20 25 14 10 11 12 15

Q2: 4 5 4 10 13

```
1 Q3 = createQueue
2 count = 0
3 loop (not empty Q1 and not empty Q2)
   1 count = count + 1
   2 dequeue (Q1, x)
   3 dequeue (Q2, y)
   4 if (y equal count)
      1 enqueue (Q3, x)
   5 end if
1 end loop
```

Part II: Applications

1- Match the following applications with the most appropriate data structure from the ones you've covered (array, linked list, stack, queue, priority queue):

a. **Social Media Feed**: This application displays a constantly updating list of posts from users you follow. New posts are added at the top, and older ones are scrolled out of view.

b. **Search Engine Results:** When you enter a search query, the engine returns a ranked list of webpages. The most relevant pages appear at the beginning of the list.

c. **Online Chat Application:** This application displays a real-time conversation between two or more users. New messages are added to the end of the conversation.

d. **Browser Back and Forward buttons:** Users can navigate back and forth through their browsing history.

e. **Music Playlist:** This application allows users to create a list of songs they want to listen to in a **specific order**. Users can remove songs at any point.

2- Stack-based App: Backtracking

One of the applications of a stack is to **backtrack**—that is, to retrace its steps.
As an example, imagine we read a list of items, such that:

- If a positive number is encountered just push it into the stack.
- Each time we read a negative number we must backtrack and print the **five** numbers that come before the negative number (starting from the one previous to the negative number) and then discard the negative number.
- If there are fewer than **five** items in the stack, print an error message and stop the program.
- The program stops normally without an error message when zero is entered.

Trace the above algorithm using the following data. Show the contents and the output screen

1 2 3 4 5 -1 1 2 3 4 5 6 7 8 9 10 -2 11 12 -3 1 2 3 4 5

3- Queue-based App : Bank Queueing System

Assume we have a bank with **one teller** working only at certain interval of time. Normally the arrival and service time for each customer should follow some pattern (usually random distribution) but for simplicity, here is the table of customer arrival and service times for each customer.

Name	Arrival Time	Service Time	Service Start Time	Service End Time	Waiting Time
A	3	12			
B	9	2			
C	16	8			
D	30	20			
E	34	10			
F	45	9			
G	47	5			

Table 1

Requirements:

1- Extend and fill the following table that shows the contents of the customers waiting queue with each enqueue or dequeue operation illustrating the time of the operation beside the queue.

Time	Operation	Waiting Queue content. Assume Front is on the left
0	Empty Queue	[ ]
3	Enqueue cust A	[A]
3	Dequeue A	[ ]

Table 2

2- Fill Table1 for each customer
3- Calculate average waiting time
4- Repeat the above problem but with two tellers in the bank and compare the new average waiting time to the old one

Part III: Implementation

1- Given linked lists classes defined as follows

```
template<typename T>
class Node
{
private :
	T item;       // A data item (can be any complex sturcture)
	Node<T>* next;      // Pointer to next node
public :
	Node( ); //default constructor
	Node( T newItem); //non-default constructor
	void setItem( T newItem);
	void setNext(Node<T>* nextNodePtr);
	T getItem() const;
	Node<T>* getNext() const;
}; // end Node

//Assume functions of class Node are fully implemented

template<typename T>
class LinkedList_Stack
{
private :
	Node<T>* head;      // Pointer to 1st node in the list
	//Add more members if needed
public :
	LinkedList_Stack( ) { head = nullptr; }
};

template<typename T>
class LinkedList_Queue
{
private :
	Node<T>* head;      // Pointer to 1st node in the list
	//Add more members if needed
public :
	LinkedList_Stack( ) { head = nullptr; }
};
```

Requirements:

A. class LinkedList_Stack is required to operates as a stack. So add only function relevant to the stack data structure (pushItem, popItem, checkEmpty)

B. class LinkedList_Queue is required to operates as a queue. So add only function relevant to the queue data structure (enqueueItem, dequeueItem, checkEmpty)

2- Implement two stacks using only one array of int. Your stack algorithm should not declare an overflow unless every slot in the array is used. You should provide 2 functions for each stack operation (e.g. Push1: pushes in stack1, Push2: pushes in stack2) and so on.

```
class doubleStack{
      enum { maxSize = 100; }
      int arr[maxSize];
      int top1, top2;
public:
      doubleStack(…….); //constructor
      …….  push1(………);
      …….  Push2(………);
      bool isFull( );
      //and so on
};
```

3- Implement queue operations using two stacks. Keeping the **FIFO** property of the queue. Assume class mySack is fully implemented with all usual stack operations

```
class myQueue{
      myStack S1;
      myStack S2;
public:
      myQueue(…….); //constructor
      …….  enqueue(………);
      …….  dequeue(………);
      //and so on
};
```